CAS Front-End Engineering

UNIT- AND INTEGRATION TESTING, TEST-DRIVEN DESIGN

Silvan Gehrig

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Lernziele

Die Vorlesung soll die Teilnehmer befähigen, automatisiertes Testing im Front-End Umfeld gezielt einzusetzen.

Die Teilnehmer können...

- **die wichtigsten Argumente für automatisiertes Testen im Front-End Umfeld erläutern.**
- **das Konzept von Test-Doubles/Mock-Objects anwenden.**
- **Test-First Design mit Jasmine einsetzen.**
- **Test-Smells erkennen und Refactorings zur Verbesserung der Tests einsetzen.**
- **die Konzepte des TDD (Test-Driven Design) erklären und die TDD-Praktiken im Grundsatz anwenden.**

Table of Contents I

- **Introduction** — **17:15 - 18:15 Lecture**
- **Test-Driven Design**
  - Basics
  - Testing with Jasmine

18:15 - 18:30 Break

18:30 - 18:45 Exercise I

- **Testing Advanced** — **18:45 - 19:30 Lecture**
  - Unit Test Patterns I

19:30 - 20:00 Break / Evening Meal

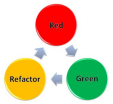
20:00 - 20:30 Exercise II

Table of Contents I

- **Testing Advanced** — **20:30 - 21:00** **Lecture**
  - Unit Test Patterns II

Exercise III / Self-Study

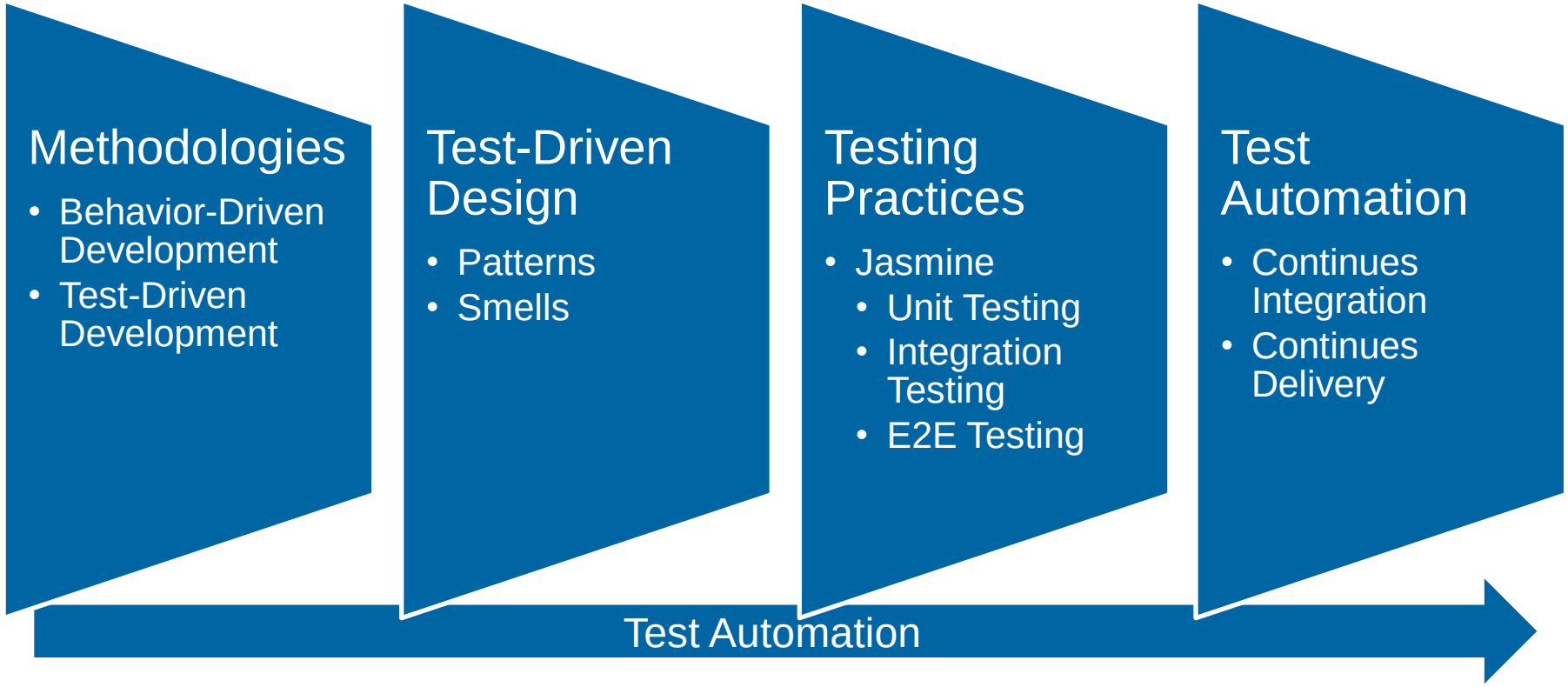
- **Test Automation** — **21:00 - 21:10** **Lecture**



TDD Katas / Workshop — **21:10 - 21:45** ***Exercise IV / Optional***

INTRODUCTION

The Big Picture



Associated principles: (not covered here)

- Manual Testing
- Regression Tests
- Acceptance Tests
- Visual Regression Tests
- Usability Testing
- Layout Tests
- Performance / Load Test
- Test Coverage / Metrics

PREFACE

INTRODUCTION

Introduction - Problem

- **Complexity of web applications is increasing**
  - Browsers accommodate more and more features
  - Applications should be running in the cloud (e.g. as a service)

- **Maintainability of web applications is a requirement today**
  - Redesign after 3 to 5 years
  - Extensions and adjustment are predictable
    - Mostly caused by changing externalities

- **High availability and stability required**
  - Business relevant applications

Introduction - Forces

- **Front End Engineering today**
  - Fragile technologies
  - Huge technology stacks
  - Nontransparent / incompatible frameworks

Engineers apply their skills often afterwards….

…too late, costs are already exploding

https://www.zdnet.de/88372991/singles-day-alibaba-macht-38-milliarden-dollar-umsatz/

Introduction - Conclusion

- **Use engineering methodologies**
  - Unit Test Patterns
  - Test-First / Test-Driven Development
- **Test your software from the beginning**
  - Automatically (on check-in / compile / daily / …)
  - UI independent
- **Choose your architecture cautiously before starting to code**
  - Layering vs Monolith

TEST-DRIVEN DESIGN

DEMO BANK ACCOUNT

TEST DRIVEN DESIGN

BASICS

TEST DRIVEN DESIGN

Basics – Test-Driven Design

- **Test anything that might break**
  - Test everything that did break (regression test)
- **New code is guilty until proven innocent**
- **Write *at least* as much test code as production code**

- **Run local tests with each compile**
- **Run all tests before check-in to repository**
  - Use Build-Guards on server-side

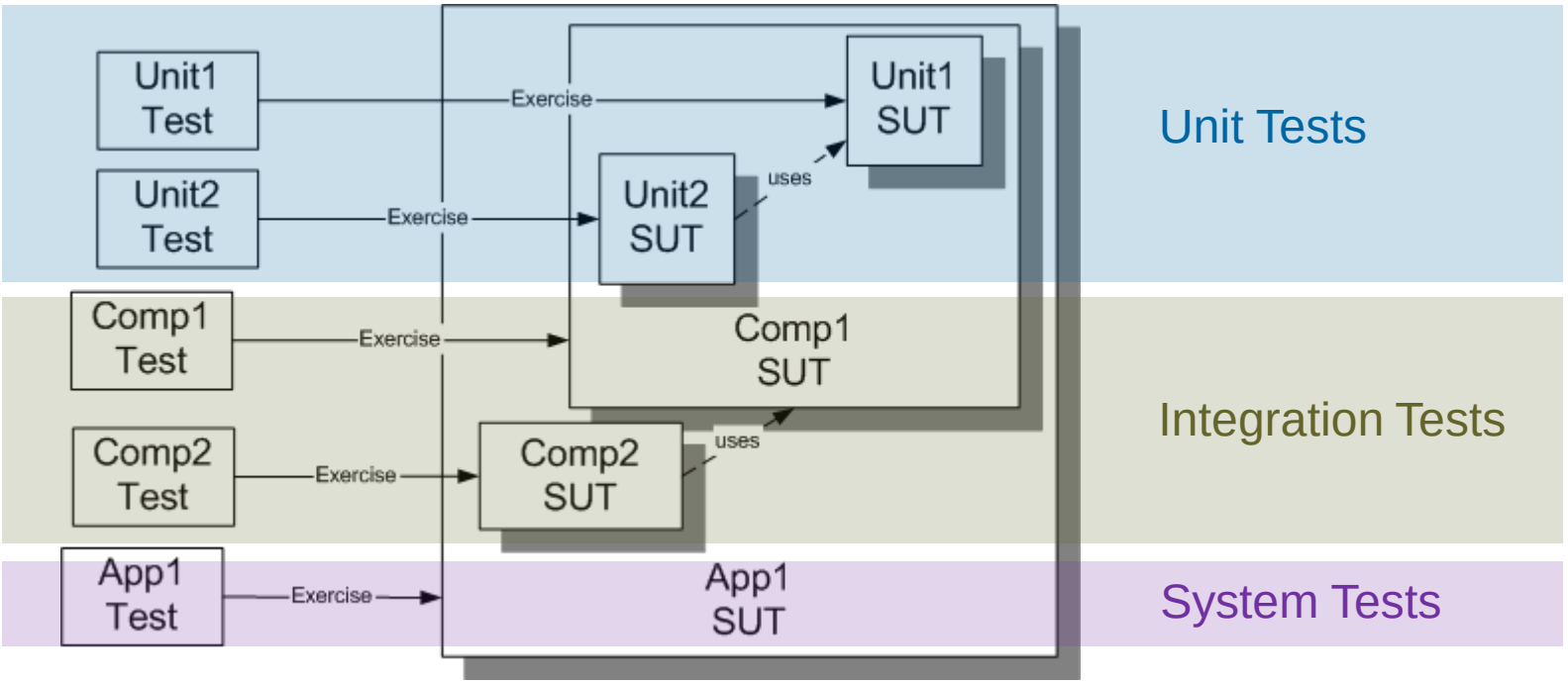
Basics – Test anything that might break

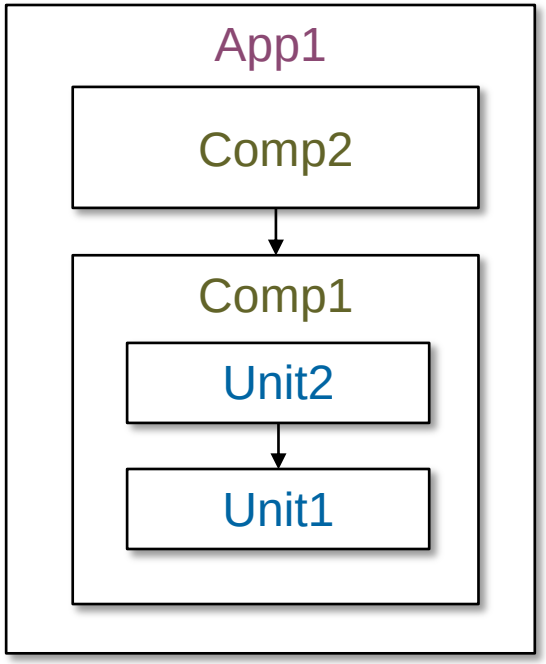
- **Why do I know that the code is working?**
- **How can I test that?**
- **What could go wrong in addition to the happy path?**
  - Is the error behavior defined?
- **How can I write tests that do not depend on external components?**

Writing good tests is hard work.

Basics – Introducing Unit Test Terminology I

- **SUT = System Under Test**

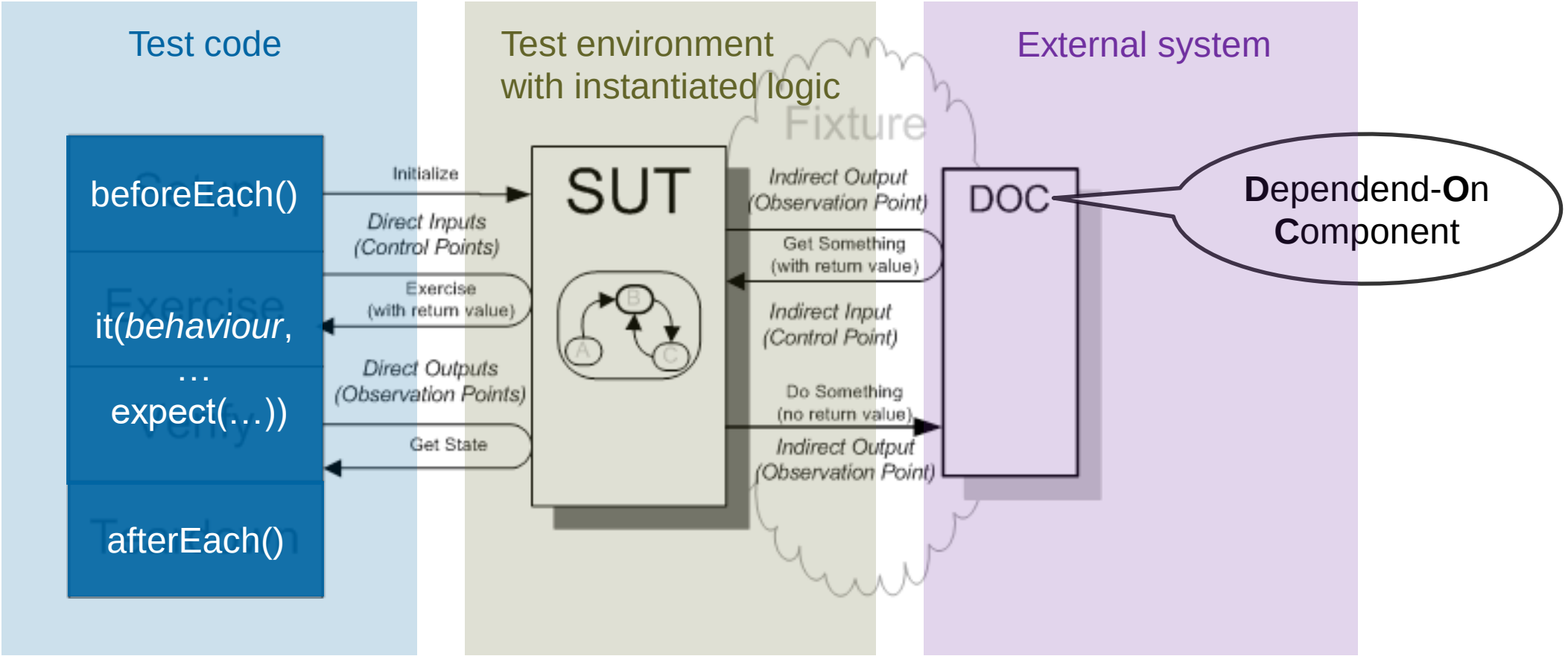




by: Gerard Meszaros [07], xUnit Test Patterns: Refactoring Test Code

Basics – Introducing Unit Test Terminology II

- **Structure of a Test Case (four phase)**



TESTING WITH JASMINE

TEST DRIVEN DESIGN

Jasmine

- **First release in August 2009**
  - Developed by Pivotal Labs (EMC Corporation / since 2015 DELL)
  - Published under MIT-License

- **Based on the ubiquitous language of Behavior-Driven Development (BDD)**
  - BDD is an extension of Test-Driven Development (TDD)
  - …relates to how the desired behavior should be specified
  - …specifies that business analysts and developers should collaborate in OOA/OOD to specify behavior in terms of user stories.

BDD Story and Specification Tools

- **Subcategory of BDD tools use specifications as an input language rather than user stories**
  - Jasmine represents a specification tool
    - doesn't use user stories as an input format
    - rather uses functional specifications for units that are being tested
    - specifications often have a more technical nature

```
BDD Specification:
Given A stack...

...then it should be empty
after created.
```

Test Specification (Jasmine)

```
describe("A stack", function() {
        it("should be empty after created.", function() {
                let a = new Stack();
                expect(a.isEmpty).toBe(true);
        });
});
```

Jasmine API

- **Provides fundamental facilities to write (BDD) tests**
  - Test suite (test fixture) begins with a call to the global Jasmine function **`describe()`**
  - Specs (test cases) are defined by calling the global Jasmine function **`it()`**
  - Expectations (assertions) are built with the function **`expect()`** which takes a value

```
describe("TestFixture", function() {
        it("Specs", function() {
                let valueToCheck = true;
                expect(valueToCheck).toBeTruthy();
        });
});
```

Test Fixture

Test Case

Assertions

More on Jasmine API

- **Provides rich set of matchers**
  - Matchers provide a boolean comparison between the actual value and the expected value.
  - Included matchers are described here
  - Custom matchers can also be implemented
    - E.g. object has a custom matcher named “toBeGoofy”.

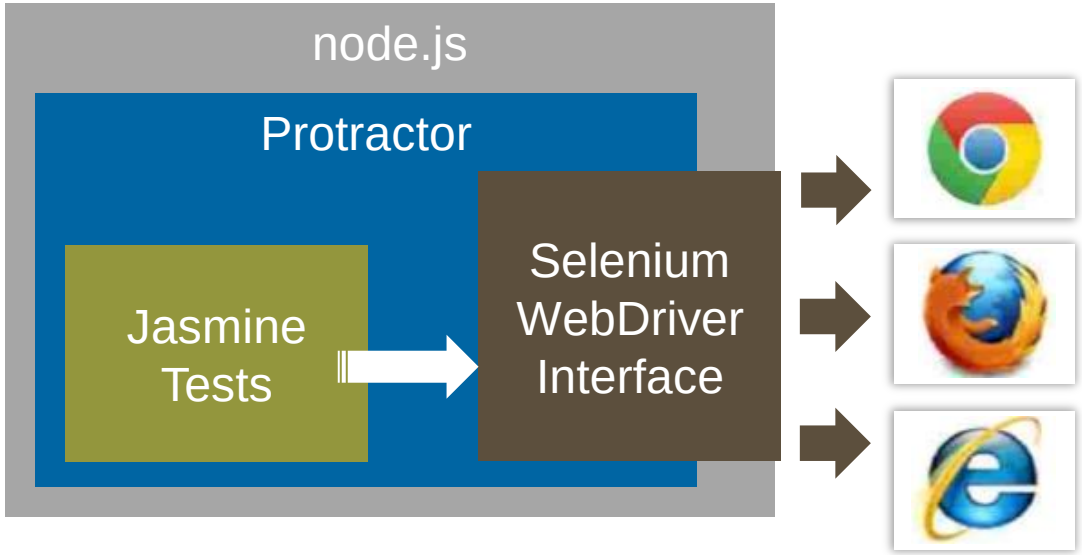
```
describe("TestFixture", function() {
        it("Specs", function() {
                let valueToCheck = true;
                expect(valueToCheck).toBeGoofy();
        });
});
```

- **Jasmine also has asynchronous specs support** **(see Asynchronous Support)**

Jasmine and the Big Picture

- **Jasmine can be used for**
  - Unit tests
  - Integration (component) tests
  - E2E (application) tests e.g. in conjunction with Protractor (often directly supported by CLI's)

- **Protractor runs tests against your application running in a real browser.**
  - …is a Node.js program built on top of WebDriverJS.
  - …uses Selenium, which accommodates multiple drivers/runners for the different browsers.



Übungsserie 1 – Working with Jasmine

15’

- **Die Übungen zum Testing verwenden Jasmine mit node.js .**
- **Die erste Übungsserie mit dem Jasmine Setup finden Sie auf https://github.com/IFS-Web/HSR.CAS-FEE.Testing/blob/master/basics/**
  - Lesen Sie die Ausgangslage sowie die Übungen durch und folgen Sie den Schritten unter *Exercise* auf dem oben angegebenen Link.
  - Die Übung bezieht sich auf die in der DEMO verwendeten Sourcen.

- **Ganz unten auf der Seite finden Sie die Lösung zur Aufgabe.**

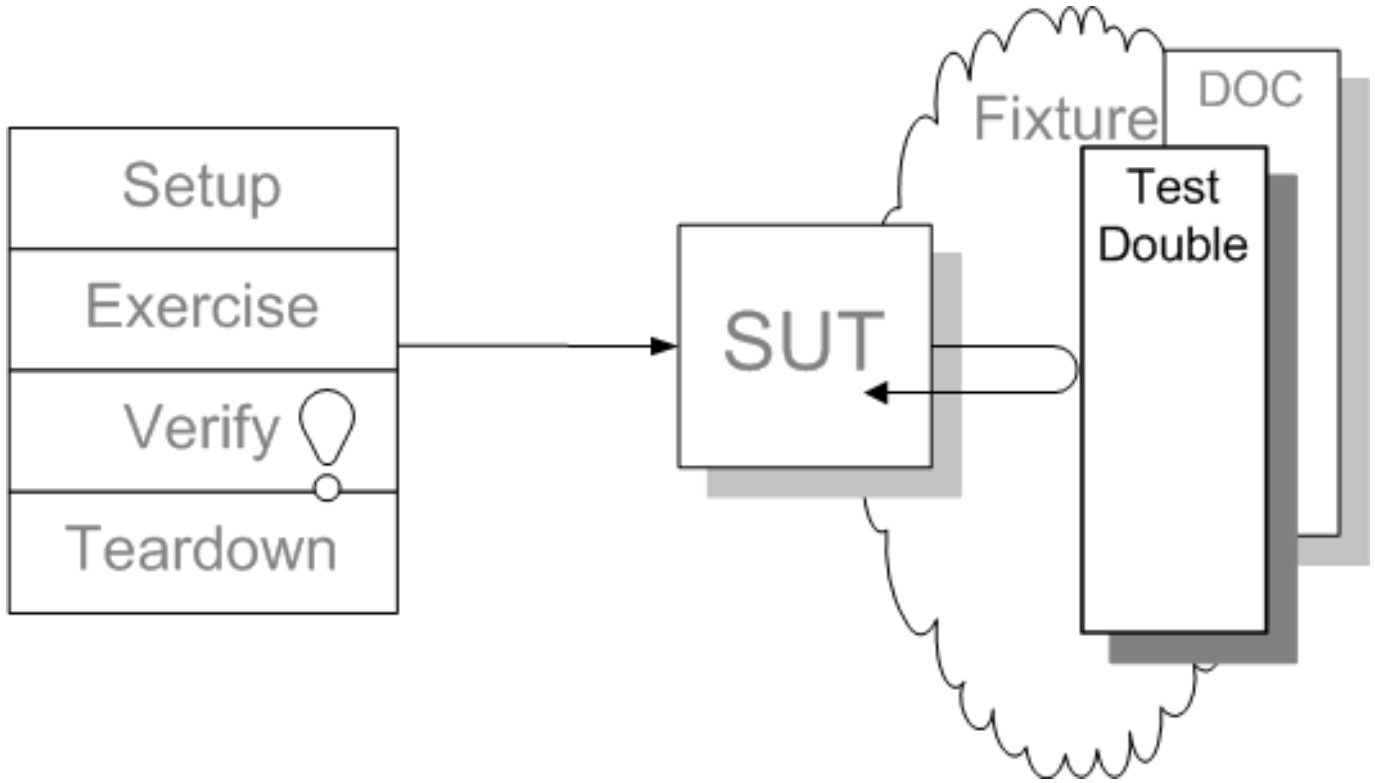
TESTING ADVANCED

UNIT TEST PATTERNS

TESTING ADVANCED

Unit Test Patterns – Test Double Pattern

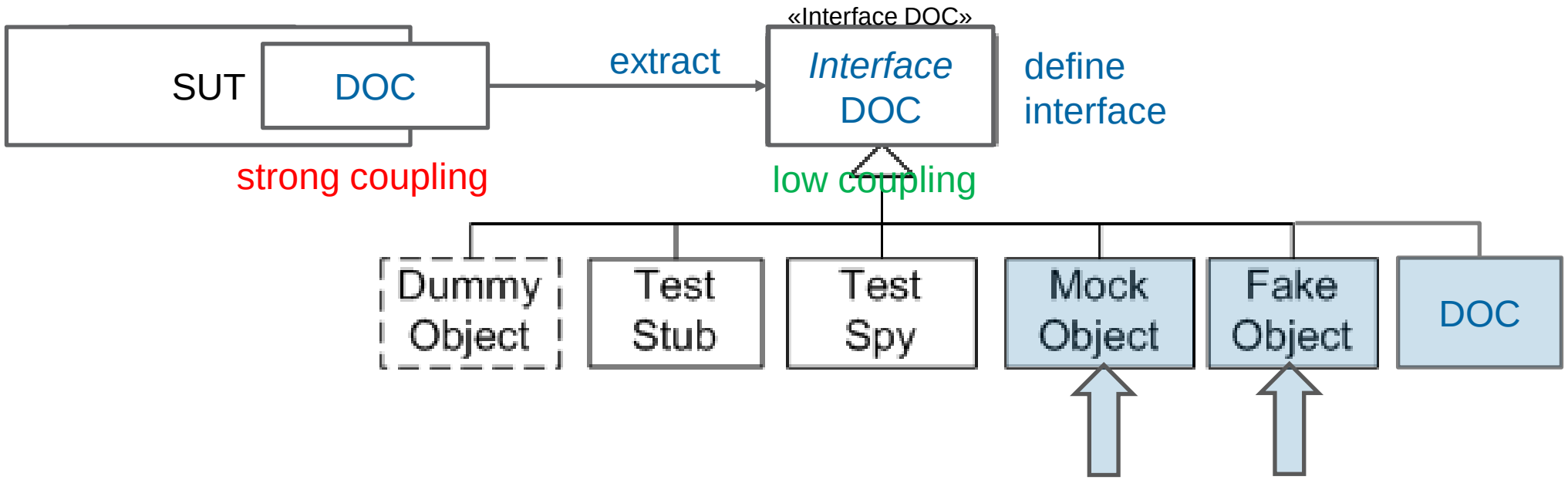
- **How can we verify logic in isolation when code it depends on is unusable?**
- **How can we avoid slow tests?**



Test Double

Unit Test Patterns – Test Double Pattern

- **How can we replace DOC with a Test Double?**



Test Double

TEST DOUBLES

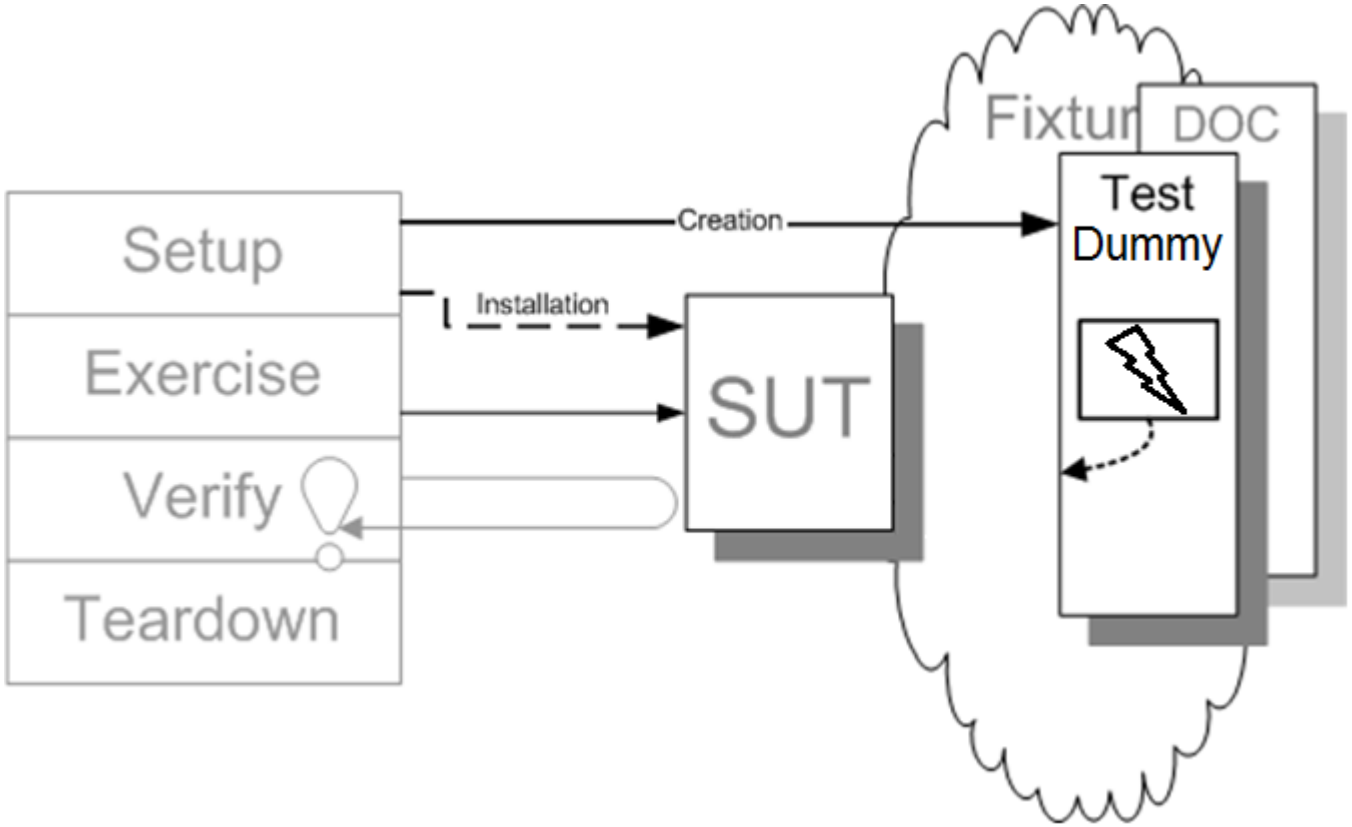
TESTING ADVANCED

Unit Test Patterns – Dummy Object Pattern

Nice to know

- **How do we specify the values to be used in tests**

  when the only usage is as irrelevant arguments of SUT method calls?

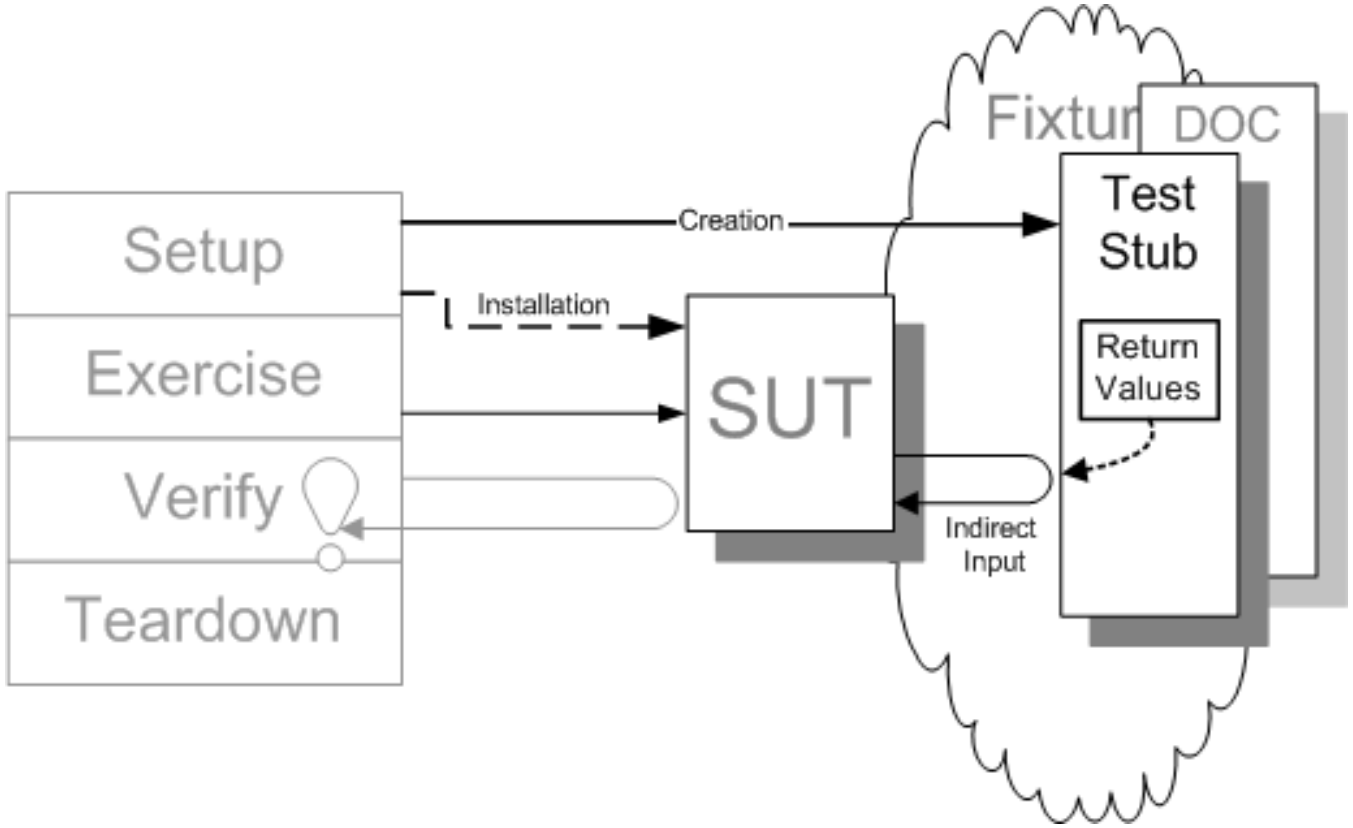


Dummy Object

Unit Test Patterns – Test Stub Pattern

Nice to know

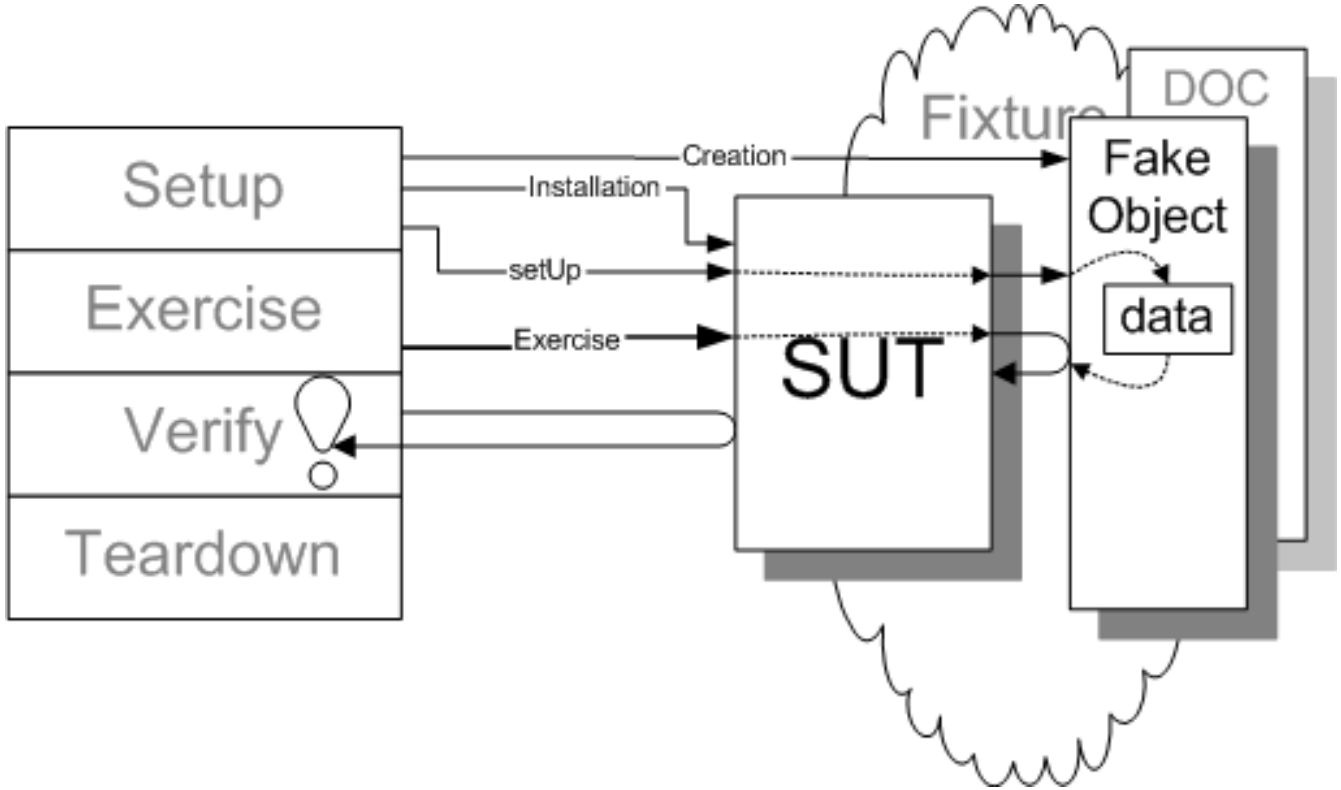
- **How can we verify logic independently when it depends on indirect inputs from other software components?**



Test Stub

Unit Test Patterns – Fake Object Pattern

- **How can we verify logic independently when depended-on objects cannot be used?**
- **How can we avoid slow tests?**

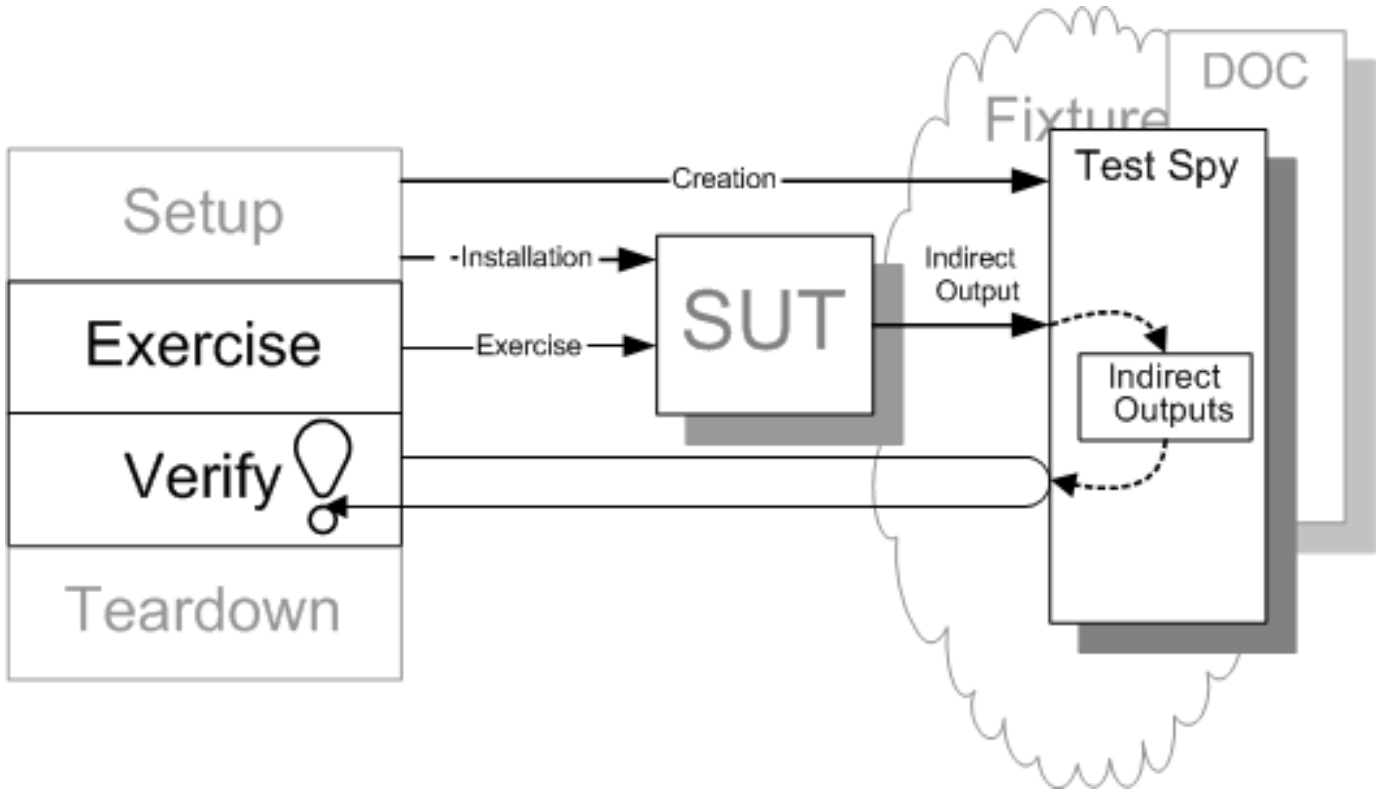


Fake Object

Unit Test Patterns – Test Spy Pattern

Nice to know

- **How can we verify logic independently when it depends on indirect inputs from other software components?**

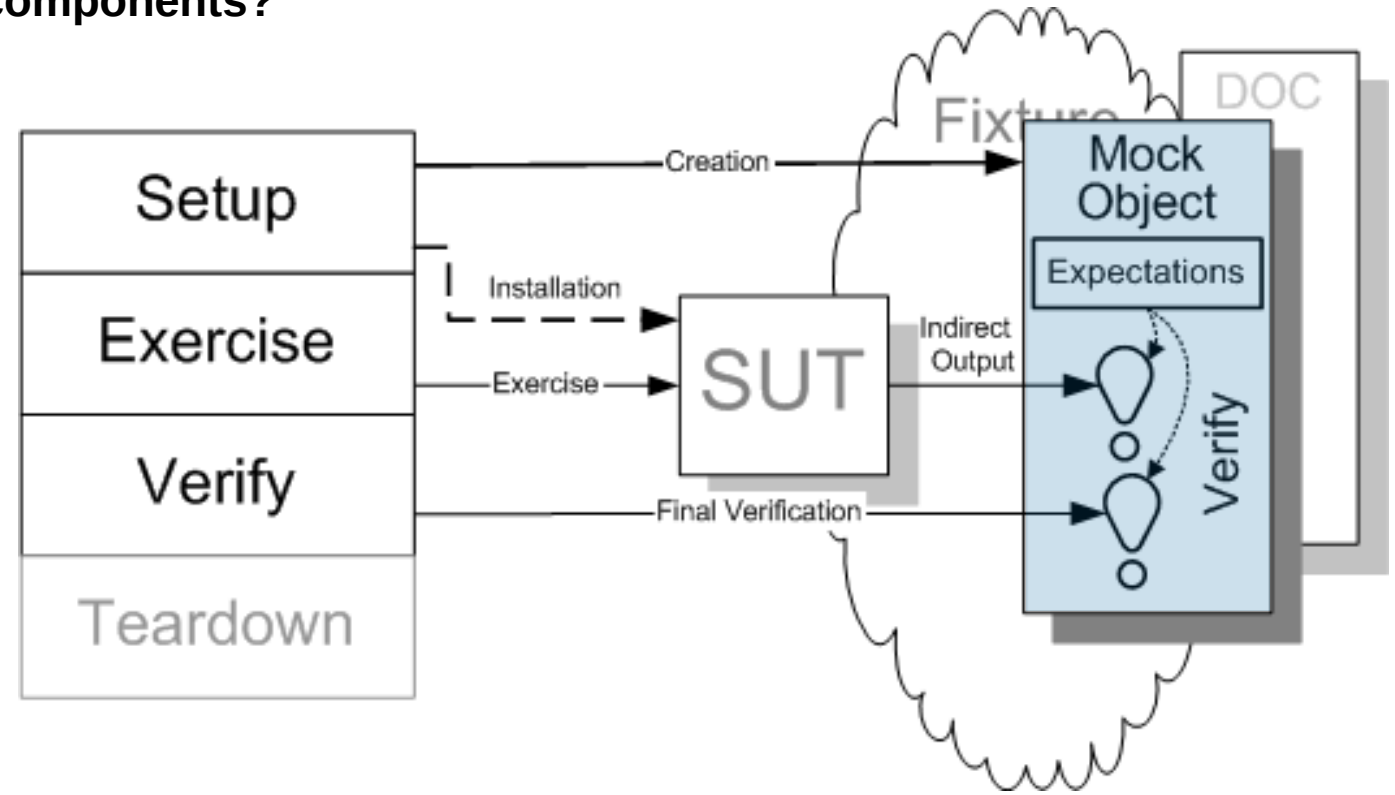


Test Spy

Unit Test Patterns – Mock Object Pattern

Similar to SPYs in Jasmine

- **How do we implement Behavior Verification for indirect outputs of the SUT?**
- **How can we verify logic independently when it depends on indirect inputs from other software components?**



Mock Object

Übungsserie 2 – Test Doubles

30’

- **Die Übungen zu den Test Doubles finden Sie auf https://github.com/IFS-Web/HSR.CAS-FEE.Testing/blob/master/dependencies/**
  - Lesen Sie die Ausgangslage sowie die Übungen durch und folgen Sie den Schritten unter *Exercise* auf dem oben angegebenen Link.
  - Verwenden Sie die Jasmine API-Informationen unter http://evanhahn.com/how-do-i-jasmine/ .

- **Ganz unten auf der Seite finden Sie die Lösung zur Aufgabe.**

UNIT TEST SMELLS

TESTING ADVANCED

Unit Test Smells – General Patterns

- **Hard-to-Test Code**
  - Poorly written code is one factor that makes it hard to write automated tests in a cost-efficient manner.
- **Production Bugs**
  - We find too many bugs during formal testing or in production.
- **Fragile Test**
  - A test fails to compile or run when the system under test (SUT) is changed in ways that do not affect the part the test is exercising.
- **Erratic Test**
  - One or more tests are behaving erratically; sometimes they pass and sometimes they fail.
- **Developers Not Writing Tests**
  - Developers aren't writing automated tests.

…even more (technology specific) Smells

- Avoid Nesting …when you're Testing

Test Smells

Unit Test Smells – Assertion Roulette

- **What's wrong here?**
  - It is hard to tell which of several assertions within the same test method caused a test failure.

```
it('should be empty when all elements are removed.', function() {
    sut.push('a');
    expect(sut.isEmpty).toBeFalsy();
    expect(sut.pop()).toBe('a');
    sut.push('b');
    sut.push('c');
    expect(sut.isEmpty).toBeFalsy();
    expect(sut.pop()).toBe('c');
    expect(sut.pop()).toBe('b');
    expect(sut.isEmpty).toBeTruthy();
  });
```

Assertion Roulette

Unit Test Smells – Test Logic in Production

- **What's wrong here?**
  - The code that is put into production contains logic that should be exercised only during tests.

```
export class Stack<T> {
  // ...
  pop(): T {
    if (!this.isEmpty) {
      return this._data.pop();
    }
    if (environment.production) {
      return null;
    } else {
      return void 0;
    }
  }
}

// ...
it('should return undefined if stack has no elements.', function() {
        expect(new Stack().pop()).toBeUndefined();
});
```

Test Logic in Production

Unit Test Smells – Obscure Test

- **What's wrong here?**
  - It is difficult to understand the test at a glance.

```
it('should return the last added element.', function() {
  sut.push('a');
  sut.push('a1');
  sut.push('b1');
  sut.pop();
  sut.push('a2');
  sut.pop();
  sut.push('b2');
  sut.pop();
  sut.pop();
  expect(sut.pop()).toBe('a');
});
```

Obscure Test

Unit Test Smells – Slow Tests

- **What's wrong here?**
  - The tests take too long to run.

```
describe('BusinessService', function() {
  let sut: BusinessService;

  beforeEach(() => {
    sut = new BusinessService(new HttpBackend('localhost:3400'))
  });

  it('should return all elements stored in the underlying data source.', function() {
    let dataFromService = sut.getData({ async: false });
    expect(dataFromService).not.toBeNull();
  });
});
```

Slow Tests

Unit Test Smells – Test Code Duplication

- **What's wrong here?**
  - The same test code is repeated many times.

```
it('should be empty when all elements are removed.', function() {
  let sut: Stack<string> = new Stack<string>();
  sut.push('a');
  sut.push('b');
  sut.push('c');
  sut.push('d');
  sut.clear();
  expect(sut.isEmpty).toBeTruthy();
});

it('should contain multiple elements.', function() {
  let sut: Stack<string> = new Stack<string>();
  sut.push('a');
  sut.push('b');
  sut.push('c');
  sut.push('d');
  expect(sut.isEmpty).toBeFalsy();
});
```

Test Code Duplication

Unit Test Smells – Conditional Test Logic

- **What's wrong here?**
  - A test contains code that may or may not be executed.

```
describe('BusinessService', function() {
  let sut: BusinessService;
  let svc: HttpBackendMock = new HttpBackendMock('example.tld');

  beforeEach(() => {
    sut = new BusinessService(svc)
  });

  it('should load and accommodate elements.', function() {
    let expectedElements: any[] = [ 'a', 'b', 'c' ];
    if (!svc.hasInMemoryData()) {
      svc.enforceData(expectedElements);
    }
    expect(sut.getData()).toBe(expectedElements);
  });
});
```

Conditional Test Logic

Übungsserie 3 – Unit Test Smells / Self-Study

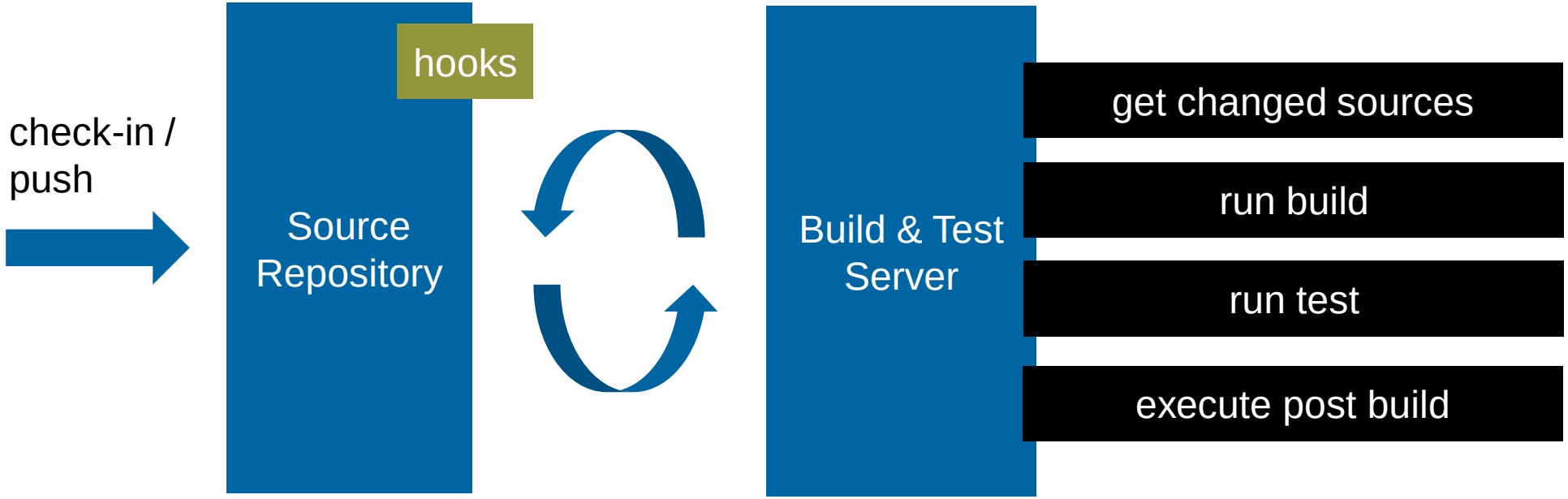
15’

- **Analysieren Sie Ihre in der Übung 2 geschriebenen Tests und vergleichen Sie Ihre Lösung mit der Lösung Ihres Banknachbarn.**
  - Welche Smells haben sich eingeschlichen?
  - Wie könnten Sie die Smells beheben?

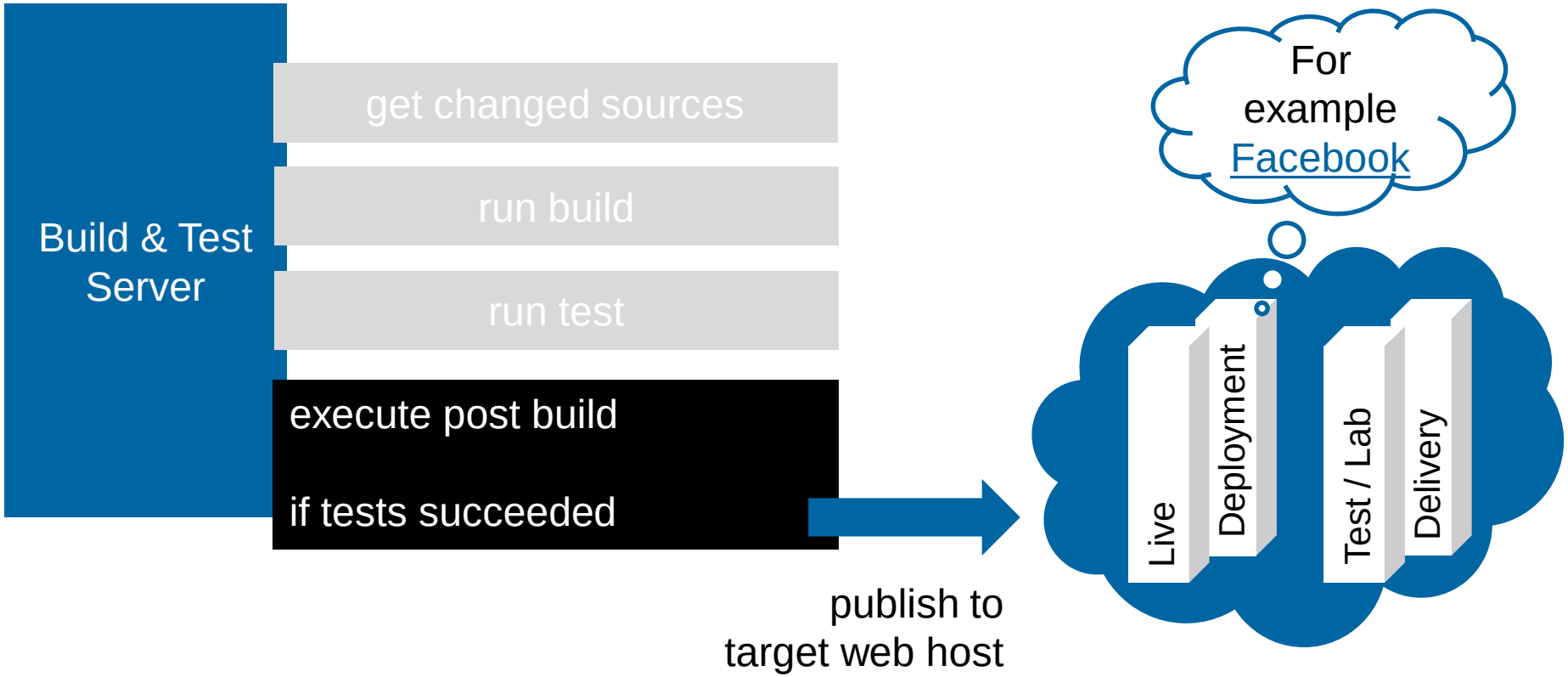
- **Schreiben Sie Ihre Tests um und korrigieren Sie die Smells.**

TEST-AUTOMATION

Test-Automation – Continuous Integration



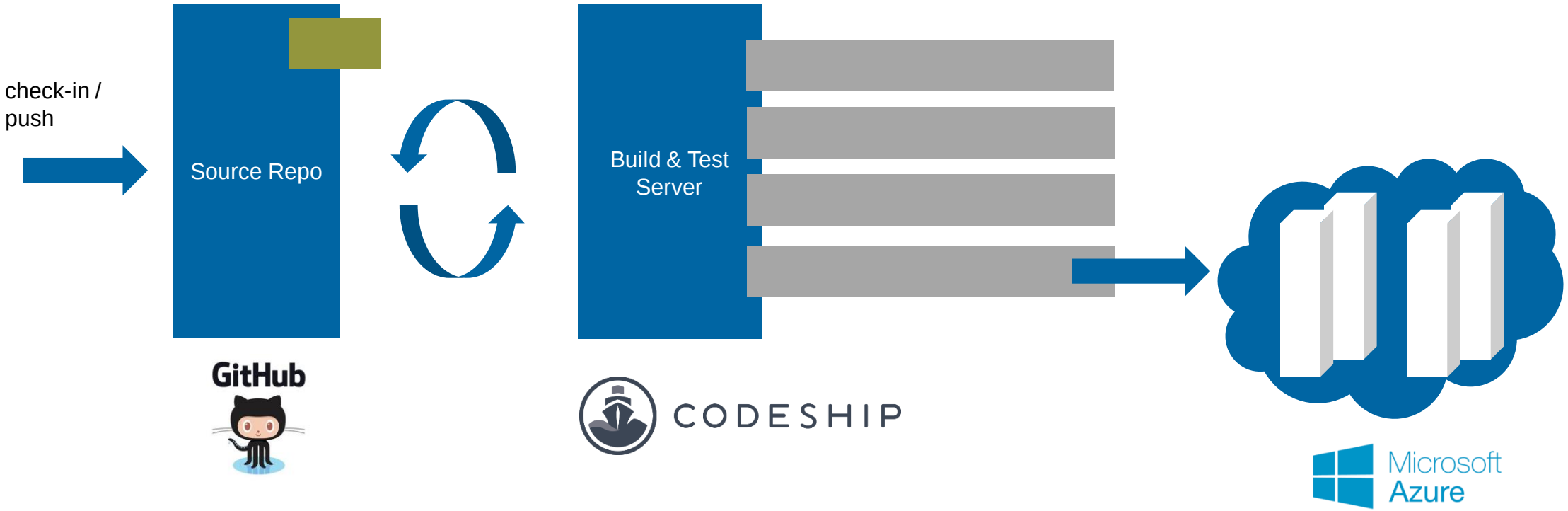
Test-Automation – Continuous Delivery/Deployment



Test-Automation – Possible Tool Chains

Test-Automation – Example Tool Chain for Angular

- **Example Tool Chain for deploying Angular Apps to the cloud**
  - See http://tattoocoder.com/angular2-azure-codeship-angularcli/



optional

PROFESSIONAL
SCRUM
DEVELOPER

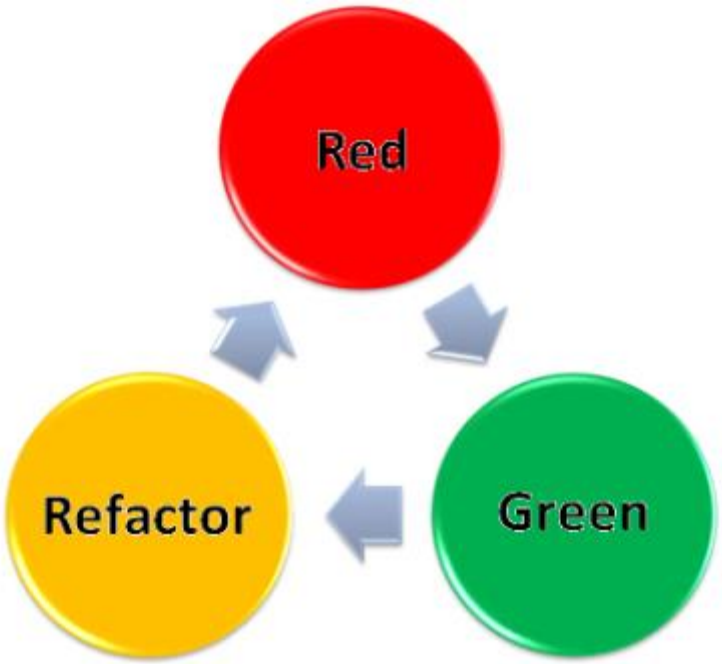
TDD KATAS / WORKSHOP

TDD - What TDD-Type are you?

- **1) Not using TDD, but curious.**
- **2) Use it sometimes. Use it when it fits.**
- **3) Cannot live without it. Use it always.**

TDD - Introduction

- **The practices of writing unit tests prior to writing the implementation code**
- **Test-Driven Development is**
  - A design practice
  - A powerful way to avoid defects in software
  - A feedback loop for validating code changes
  - A way to write unit tests



Übungsserie 4a – Test-Driven Development

10 min

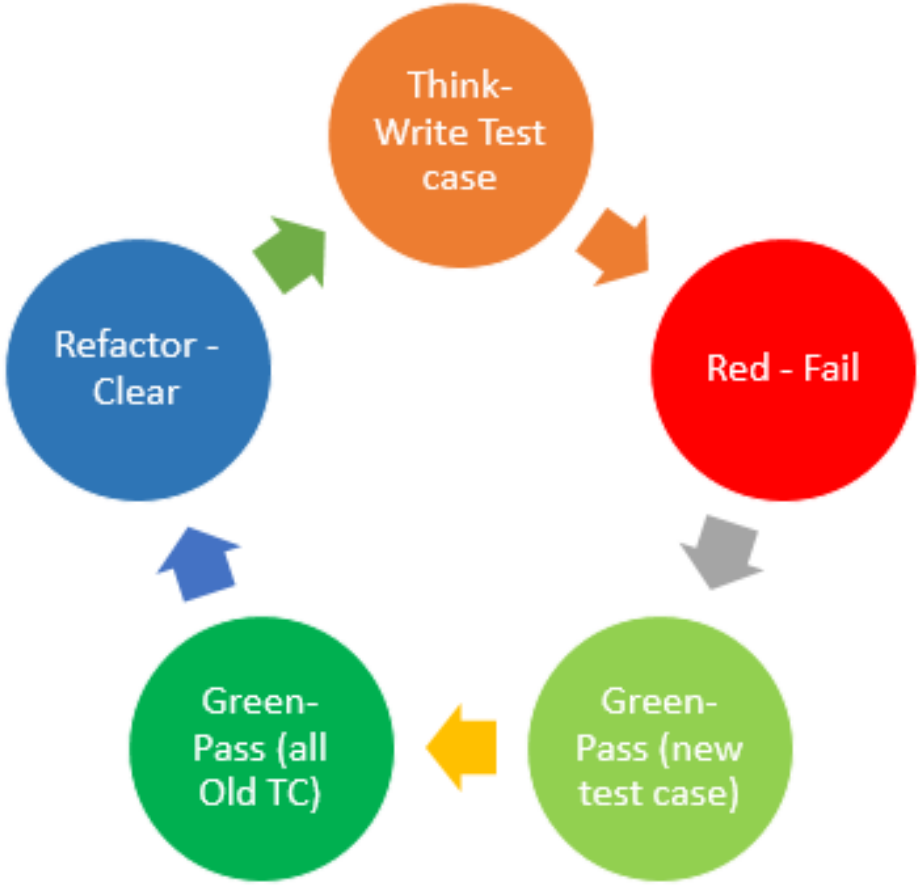
- **Work within your node.js and Jasmine environment cloned from GitHub.**
- ***Objective:* Create a simple String calculator.**

- **Requirements**
  - The calculator must take 0, 1 or 2 numbers,
    and return their sum,
    for example “” or “1” or “1,2”.
  - For an empty string it will return 0.
  - Return values are Numbers.

Übungsserie 4a – Test-Driven Development

- **Who has written a class `Calculator`?**
- **How many tests do you have?**
- **How many times did you run your tests?**
- **Did you refactor your code?**

TDD – The Cycle



Übungsserie 4b – New Requirements

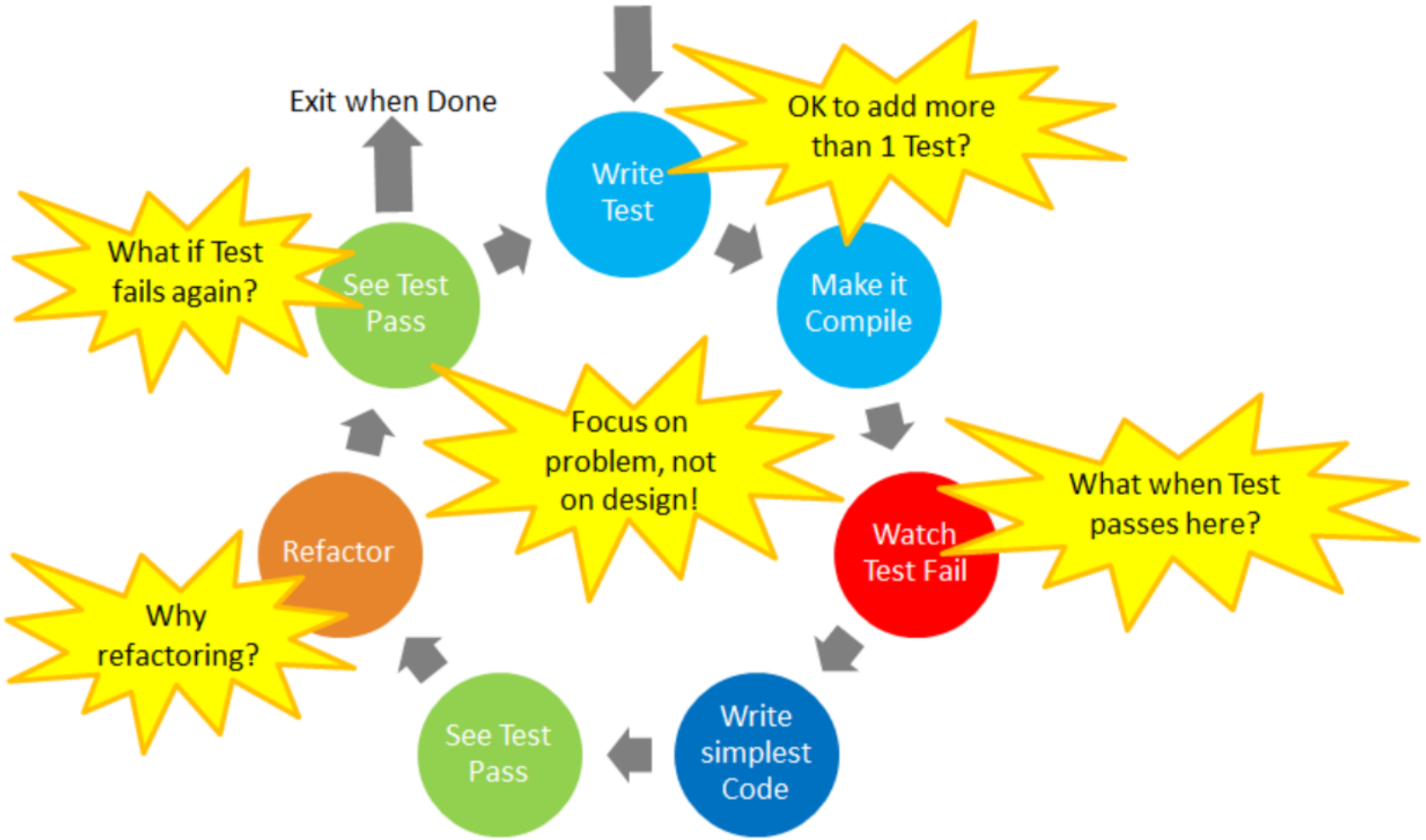
10 min

- **Allow adding 3 numbers.**
- **Adding 4 and more numbers throws an error “illegal argument exception”.**
- **Allow to handle new lines between numbers (in addition to commas).**
  - “1\n2,3” will equal 6
  - “2,4,4” will equal **9**
  - “3\n4\n5” will equal 12

Übungsserie 4b – Test-Driven Development

- **Who followed the TDD cycle strictly?**
- **Who has written more than 1 test at once?**
- **How did you test if exception was thrown?**

TDD – The Cycle



Übungsserie 4c – New Requirements

(optional)

- **Results should be stored to `localStorage` in a history.**

  Pseudo code for node.js:

```
const LocalStorage = require('node-localstorage').LocalStorage;
const localStorage = new LocalStorage('./calculator');

// read
let history = JSON.parse(localStorage.getItem('CalculationResult') || '[ ]' );

// write
localStorage.setItem('CalculationResult', JSON.stringify( [ …history, 5 /* new-result */] ));
```

Übungsserie 4c – Storage

- **How did you test writing to `localStorage`?**
  - Did you read from the `localStorage`?
  - Did you inject a `LocalStorageService`?

- **If reading directly from `localStorage`: Is this good?**

QUESTIONS?

Sources

- **Slides**
  - Prof. Peter Sommerlad, HSR Rapperswil, Software Engineering Slides
  - Prof. Hans Rudin, HSR Rapperswil, Software Engineering Slides
  - Mirko Stocker, HSR Rapperswil, Continuous Integration / Delivery Concepts
  - Daniel Tobler, Zühlke Engineering, TDD Katas

- **Books**
  - Gerard Meszaros, xUnit Test Patterns - Refactoring Test Code ISBN-13: 978-0131495050
  - Christian Johansen, Test-Driven JavaScript Development ISBN-13: 978-0321683915
  - Evan Hahn, JavaScript Testing with Jasmine ISBN-13: 978-1449356378
  - Kent Beck, Test Driven Development: By Example ISBN-13: 978-0321146533

Sources

- **Web Resources**
  - http://jasmine.github.io/
  - http://tattoocoder.com/angular2-azure-codeship-angularcli/
  - http://xunitpatterns.com/
  - https://dannorth.net/introducing-bdd/
  - https://en.wikipedia.org/wiki/Behavior-driven_development
  - https://github.com/angular/protractor/
  - https://github.com/SeleniumHQ/selenium/wiki/WebDriverJs/